Agentenbasierte Modellierung und Simulation schwarmintelligenter Algorithmen am Beispiel der Futtersuche von Ameisen und Bienen

Studienarbeit

für die Prüfung zum

Bachelor of Engineering

des Studienganges Informationstechnik

an der

Dualen Hochschule Baden-Württemberg Karlsruhe

von

Tim Saupp

Abgabedatum 18.05.2018

Bearbeitungszeitraum	30.10.2017-18.05.2018
Matrikelnummer	2742603
Kurs	TINF15B3
Gutachter der Studienakademie	Prof. Dr. Lausen

Erklärung

Ich versichere hiermit, dass ich meine Studienarbeit mit dem Titel: Agentenbasierte Modellierung und Simulation schwarmintelligenter Algorithmen am Beispiel der Futtersuche von Ameisen und Bienen selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Ort Datum

Unterschrift

Sperrvermerk

Der Inhalt dieser Arbeit darf weder als Ganzes noch in Auszügen Personen außerhalb des Prüfungsprozesses und des Evaluationsverfahrens zugänglich gemacht werden, sofern keine anders lautende Genehmigung der Ausbildungsstätte vorliegt.

Zusammenfassung

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ABM Agentebasierte Modellierung

ACO Ameisenkolonieoptimierung

BCO Bienenkolonieoptimierung

PSO Partikelschwarmoptimierung

1 Projektbeschreibung

1.1 Motivation

1.2 Ziel der Arbeit

Ziel der Studienarbeit ist die agentenbasierte Modellierung und Simulation der Futtersuche von Ameisen und Bienen. Dazu werden die Strategien der schwarmintelligenten Algorithmen ACO und BCO auf das Modell der Futtersuche angewandt. Als abstraktes Beispiel wird zudem die PSO zum Finden eines globalen Minimums der Rastrigin-Funktion implementiert.

1.3 Kapitelübersicht

Kapitel 2 erläutert den Begriff schwarmintelligente Superorganismen, definiert den Begriff Schwarmintelligenz und beschreibt die Grundlagen der agentenbasierte Modellierung. In Kapitel 3 werden die schwarmintelligenten Algorithmen PSO, ACO und BCO vorgestellt. Die implementierten Modelle werden in Kapitel 4 definiert. Zudem wird die Auswahl des verwendeten Frameworks zur Implementierung der definierten Modelle gerechtfertigt. Nachfolgend wird in Kapitel 5 die Implementierung der agentenbasierten Modelle aus Kapitel 4 beschrieben beschrieben.

2 Grundlagen

2.1 Schwarmintelligente Superorganismen

Durch die sensuale Verbindung der Tiere wird die Futtersuche und das Abwehren von Gefahren ohne eine zentrale Lenkung bzw. ohne hierarchische Befehlskette bewältigt. Instinktiv verankerte Regeln sorgen dafür, dass auf bestimmte Aktionen der Tiere in vollkommen deterministischer Weise eine Reaktion erfolgt. Aus dieser dezentralen Interaktion entstehen, bei einem Kollektiv von Tieren, intelligente Resultate auf Makroebene.

2.1.1 Begriffsdefintion

Bereits 1911 bezeichnet W. M. Wheeler, amerikanischer Ethologe mit Spezialisierung auf dem Gebiet der Erforschung sozialer Insekten, Kolonien wie die der Bienen und Ameisen als Superorganismen mit emergenten Fähigkeiten[1]. Abbildung 1 zeigt die Definition und Entstehungsbedingungen kollektiver Intelligenz bei Tierschwärmen:

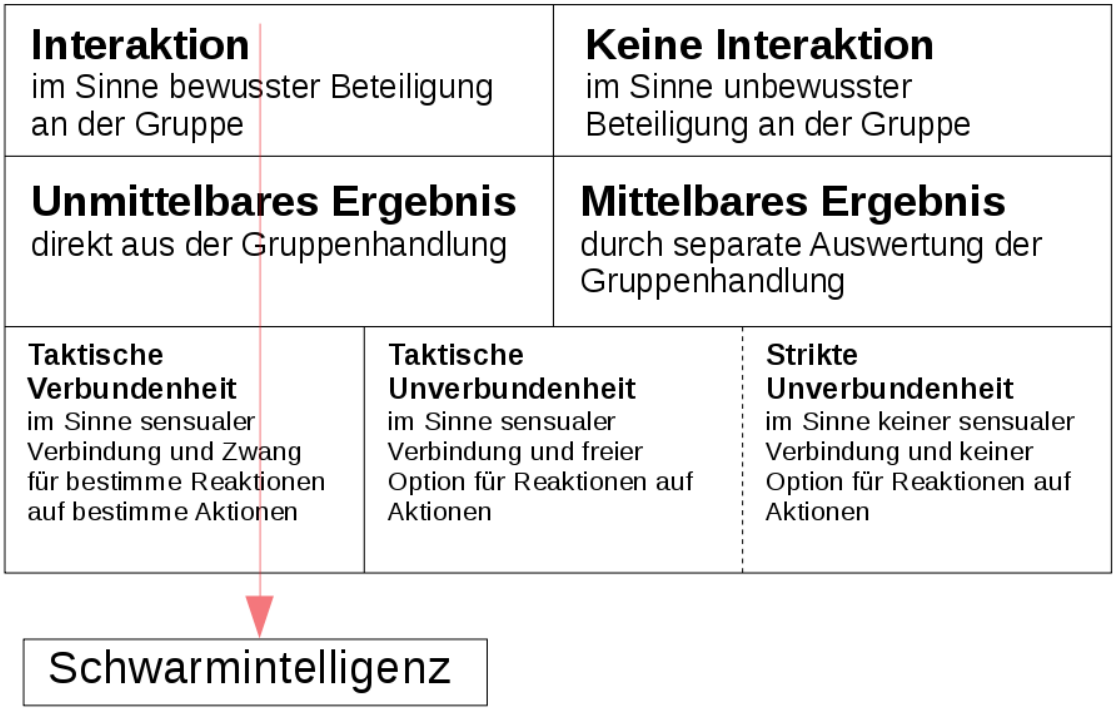

Abbildung 1: Entstehungsbedingung und Definition kollektiver Intelligenz bei Tierschwärmen[3]

- **Interaktion:** Auf Aktion und Reaktion basierende Interaktion gilt als Grundlage für die Definition des Begriffs Schwarmintelligenz. Anlass dafür ist der in den Tieren vorhandene Überlebensinstinkt, der diese veranlasst, sich bewusst an der Gruppe zu beteiligen.
- **Unmittelbares Ergebnis:** Einzelne Tiere führen Handlungen aus ohne Wissen um das Schwarmergebnis. Das Resultat entsteht unmittelbar aus der Handlung des Schwarms und bedarf keiner externen Aggregation und Auswertung.

[1] W.M. Wheeler (1911)
[3] A. Aulinger (2013)

- **Taktische Verbundenheit:** Sowohl die sensuale Verbindung der Tiere als auch der in den Tieren verankerte Instinkt zeugen von der taktischen Verbundenheit des Schwarms bestehend aus festen Aktions- und Reaktionsmustern.

Zusammenfassend beschreibt der Begriff Schwarmintelligenz ein Phänomen aus dem Tierreich zur Selbstorganisation eines Schwarms, um lebensnotwendige Aufgaben gemeinsam, auf intelligente Weise, zu bewältigen. Dabei vollbringen die Tiere im Schwarm Leistungen, die das Vermögen jedes Einzeltiers übersteigen.

2.1.2 Ameisen

Ein Ameisenstaat passt sich ständig an neue Gegebenheiten seiner Umwelt an. Ameisen bilden Staaten mit einigen hundert bis zu mehreren Millionen Individuen. Trotz dieser riesigen Anzahl funktioniert ein Ameisenstaat, da er sich selbst organisiert ohne eine hierarchische Instanz, die einen Überblick über alle Aufgaben besitzt oder diese steuert und verteilt. Stattdessen führen die Handlungen einzelner Ameisen im Zusammenspiel zu einem organisierten Staat, der für die Ameisen sorgt und Nahrung, Brutpflege und Schutz bietet. Interessant sind die Ameisen aufgrund ihrer Fähigkeit effiziente Wege zwischen Futterquellen und dem Ameisenbau ausfindig zu machen[4].

Ist der Futtervorrat des Ameisenbaus erschöpft verlassen mehrere Ameisen den Ameisenbau gleichzeitig und begeben sich auf die Futtersuche. Sobald eine Ameise eine Futterquelle gefunden hat, nimmt sie eine Gewichtseinheit des Futters mit und begibt sich auf den Rückweg zum Ameisenbau. Dabei setzt die Ameise Pheromone frei die mit der Zeit verfliegen, um den Weg zur Futterquelle zu markieren. Die Ameise die den kürzesten Weg zu einer Futterquelle gefunden hat legt die Strecke zwischen Ameisenbau und Futterquelle häufiger zurück. Die Pheromonspur wird durch das häufige Zurücklegen der Strecke intensiviert und dient als sicherer Wegweiser zur Futterquelle. Mitglieder der Kolonie folgen den intensivsten Pheromonspuren. Ist die Futterquelle erschöpft, löst sich die Pheromonspur auf[5].

2.1.3 Bienen

Um den Gesamtenergiebedarf des Schwarms zu ermitteln, orientiert sich die einzelne Sammlerin an der Wartezeit bei der Übergabe des gesammelten Nektars an die Bienen die den Nektar speichern: Je voller die Futterspeicher, desto länger müssen die Speicherbienen nach leeren Zellen suchen. Je leerer die Speicher, desto schneller wird die Abgabe des Nektars abgewickelt.

Im kilometerweiten Gelände besitzt keine Biene den gesamten geographischen Überblick. Sie entscheidet nur lokal über die Rentabilität der Futterquelle. Kundschafterinnen und Sammlerinnen, die von der Futtersuche wiederkehren, führen einen Tanz auf und zeigen unbeschäftigten Bienen damit die Richtung und Entfernung zur gefundenen Futterquelle. Unbeschäftigte Bienen sehen sich die Tänze der im Bienenstock eintreffenden Bienen an und entscheiden sich anschließend für ihr nächstes Ziel. Sobald eine Futterquelle erschöpft ist brauchen die Sammlerinnen länger beim Sammeln und veranlassen aufgrund der geringeren Anzahl an Tänzen weniger Bienen dazu am gleichen Ort zu sammeln.

[4] L. Pintscher (2008)
[5] R. Wehner (2001)

2.2 Agentenbasierte Modellierung

Die Agentebasierte Modellierung (ABM) erlaubt eine natürliche Beschreibung von Systemen als eine Sammlung autonomer entscheidungsfähiger Agenten in einer gemeinsamen Umgebung, um emergente Phänomene zu analysieren. Das Ziel der ABM besteht darin, durch die Simulation einer Vielzahl an Agenten, das resultierende Systemverhalten zu untersuchen. Nach C. Macal und M. North[6] verfügt ein agentenbasiertes Modell über folgende Komponenten:

- **Agenten:** Agenten handeln autonom, proaktiv und reaktionär auf der Basis von festgelegten Regeln. Jeder Agent besitzt nur eine eingeschränkte Sicht auf das Gesamtsystem. Sein Wissen über den globalen Zustand des Systems ist immer unvollständig.
- **Agenten-Beziehungen:** Beziehungen zwischen Agenten entstehen durch die proaktive Aktion eines Agenten und die darauffolgende Reaktion eines anderen Agenten oder Interaktionen der Agenten mit der Umgebung.
- **Agenten-Umgebung:** Agenten interagieren innerhalb einer Umgebung mit anderen Agenten und ihrer Umgebung.

2.2.1 Agenten

M. Wooldridge und N. Jennings[7] weisen Agenten die Charaktereigenschaften Autonomie, Proaktivität, Reaktivität und die Fähigkeit zur Interaktion durch Kommunikation zu. Die Handlungsautonomie eines Agenten beschränkt sich auf die Fähigkeit, eigenständig zu entscheiden, welche der ihm zur Verfügung stehenden Aktionen situationsbedingt auszuführen ist. Zielvorgaben bestimmen wann ein Agent welche Aktionen ausführt. Proaktivität beschreibt dabei das zielgerichtete Verhalten eines Agenten der selbst aktiv wird, statt nur auf die Umgebung zu reagieren.

2.2.2 Agenten-Beziehungen

Zwischen den Systemelementen bestehen Interaktionsbeziehungen. Die technischen Voraussetzungen für die Kommunikation werden von der Umgebung bereitgestellt. Beziehungen lassen sich einteilen in Interaktion zwischen Agenten und Umgebung, Interaktion zwischen Agenten und organisatorisch bedingte Beziehungen. Die Kommunikation zwischen Agenten kann dabei direkt durch Austausch von Nachrichten oder indirekt über die Veränderung der Umgebung erfolgen.

2.2.3 Agenten-Umgebung

Nach S. Russell und P. Norvig[8] muss eine geeignete Agenten-Umgebung zugänglich, deterministisch, dynamisch, kontrollierbar und teleologisch sein. Eine zugängliche Umgebung erlaubt den Agenten Zugriff auf ihren Zustand. In der Regel beschränkt sich dieser Zugriff jedoch auf den lokalen Wahrnehmungsbereich eines Agenten und erlaubt somit nur Zugriff auf einen Ausschnitt der Umgebung. Deterministisch ist die Umgebung, sobald der Folgezustand vollständig durch den aktuellen Umgebungszustand und die aktuelle Aktion des Agenten bestimmt ist. Kann sich der Zustand der Umgebung durch Aktionen der Agenten ändern, so handelt es sich um eine dynamische und kontrollierbare Umgebung.

[6] C. Macal, M. North (2010)
[7] M. Wooldridge, N. Jennings (1995)
[8] S. Russel, P. Norvig (2009)

3 Schwarmintelligente Algorithmen

3.1 Particle Swarm Optimization

Die klassische Partikelschwarmoptimierung (PSO) wurde erstmals im Jahr 1995 von J. Kennedy und R. Ebert beschrieben. Die PSO stellt ein, auf der Abfolge von abstrakten Schritten basiertes, Verfahren zur näherungsweisen Lösung von Optimierungsproblemen dar. Zur Lösung des Optimierungsproblems wird eine Population von Partikeln, so lange durch einen Suchraum bewegt, bis eine hinreichende Lösung in einer bestimmten Zeit gefunden wird. Dabei stellt die Position eines Partikels eine potentielle Lösung dar und wird in jedem Zeitschritt neu berechnet[9]. Um die PSO auf ein mathematisches Problem anwenden zu können, müssen zunächst der Suchraum und die Partikel definiert werden[10].

3.1.1 Suchraum

Der Suchraum $S \subset \mathbb{R}^n$ ist ein n-dimensionaler Raum, in dem sich die gesamte Population $X = \{x_1, x_2, ..., x_n\}$ der Partikel bewegt. Damit eine Lösung gefunden werden kann beschränkt sich der Suchraum auf eine endliche Anzahl an möglichen Positionen, die von den Partikeln besucht werden können.

3.1.2 Partikel

Jeder Partikel x_i besitzt folgende Eigenschaften im Suchraum S zum Zeitpunkt t:

- **Position $x_i(t)$ im Suchraum S mit einem Funktionswert $f(x_i(t))$:** Die aktuelle Position des Partikels x_i zum Zeitpunkt t repräsentiert eine potentielle Lösung des Optimierungsproblems. Der dazugehörende Funktionswert $f(x_i(t))$ beschreibt die Güte der potentiellen Lösung.
- **Position $x_{b,i}(t)$ im Suchraum S mit einem Funktionswert $f(x_{b,i}(t))$:** Die Position $x_{b,i}$ ist die bisher beste Position des Partikels x_i zum Zeitpunkt t.
- **Position $x_g(t)$ im Suchraum S mit einem Funktionswert $f(x_g(t))$:** Die Position x_g ist die bisher beste Position des Partikelschwarms zum Zeitpunkt t.
- **Geschwindigkeit $v_i(t)$:** Die Geschwindkeit v_i mit der sich der Partikel durch den Suchraum S zum Zeitpunkt t bewegt.
- **Trägheitskoeffizient w:** Der Trägheitskoeffizient w bestimmt die Gewichtung der Geschwindigkeit $v_i(t)$ zum Zeitpunkt $t + 1$.
- **lokaler Vertrauenskoeffizient a_l:** Der lokale Vertrauenskoeffizient a_l bestimmt die Gewichtung der lokalen Attraktion, bei der Berechnung der Geschwindigkeit $v_i(t + 1)$.
- **globaler Vertrauenskoeffizient a_g:** Der globale Vertrauenskoeffizient a_g bestimmt die Gewichtung der globalen Attraktion, bei der Berechnung der Geschwindigkeit $v_i(t + 1)$.

Der Trägheitskoeffizient w und die Vertrauenskoeffizienten a_l und a_g geben an, wie sehr die Partikel sich selbst, ihren eigenen Erfahrungen und den Erfahrungen ihrer Nachbarn vertrauen.

[9] Y. Liu (2014)
[10] E. Konstantinos, N. Michael (2010)

3.1.3 Algorithmus

Initialisiere Partikelschwarm X im Suchraum S;
while *Maximale Iterationen nicht erreicht* **do**
  Bestimme für jeden Partikel Position $x_i(t)$;
  Bestimme für jeden Partikel Geschwindigkeit $v_i(t)$;
  Berechne für jeden Partikel den Funktionswert $f(x_i(t))$;
  if $f(x_i(t))$ *besser als* $f(x_{b,i}(t))$ **then** Setze $x_{b,i}(t) = x_i(t)$;
  if $f(x_i(t))$ *besser als* $f(x_g(t))$ **then** Setze $x_g(t)) = x_i(t)$;
  Berechne neue Geschwindigkeit $v_i(t+1)$ des Partikels x_i;
  Berechne neue Position $x_i(t+1)$ des Partikels x_i;
end

Algorithm 1: PSO Algorithmus

Der abgebildete Algorithmus zeigt den Ablauf der klassischen PSO. Im ersten Schritt wird die Population X mit zufälligen Positionen x_i im Suchraum S initialisiert. Bis die Anzahl an maximalen Iterationen nicht erreicht ist wird für jeden Partikel geprüft ob der Funktionswert $f(x_i(t))$ für die Position $x_i(t)$ besser ist als der Funktionswert $f(x_i(t-1))$ im vorherigen Schritt an der Position $x_i(t-1)$. Ist der Funktionswert $f(x_i(t))$ für die Position $x - i(t)$ des Partikels besser, wird die bisher beste Position des Partikels $x_{b,i}(t)$ zu $x_i(t)$. Zudem wird für jeden Partikel x_i geprüft, ob der aktuelle Funktionswert $f(x_i(t))$ des Partikels besser ist als der globale Funktionswert $f(x_g(t))$. Ist dies der Fall so wird die global beste Position $x_g(t)$ zu $x_i(t)$. Für die nächste Iteration wird die Geschwindigkeit $v_i(t+1)$ und die neue Position $x_i(t+1)$ des Partikels nach folgenden Formeln berechnet:

$$v_i(t+1) = wv_i(t) + a_l r_1(x_{b,i}(t) - x_i(t)) + a_g r_2(x_g(t) - x_i(t)) \tag{1}$$

$$x_i(t+1) = x_i(t) + v_i(t+1) \tag{2}$$

Bei der Berechnung der neuen Geschwindigkeit $v_i(t+1)$ wird die Entfernung der Position $x_g(t)$ zur Position $x_{b,i}(t)$ berechnet. Dabei fallen die Zufallszahlen $r_1, r_2 \subset \mathbb{R}$ ins Gewicht, sowie die Parameter w, a_l, a_g, mit denen das Verhalten des Schwarms verändert wird. Gilt $a_l > a_g$, dann streben die Partikel in Richtung von $x_{b,i}(t)$. Ist $a_g > a_l$, dann konvergieren die Partikel zur Position $x_g(t)$. Abbildung 2 zeigt in grafischer Darstellung die Berechnung der neuen Geschwindigkeit $v_i(t+1)$ und der neuen Position $x_i(t+1)$.

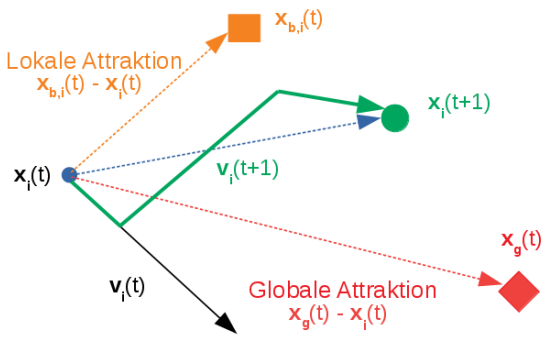

Abbildung 2: Berechnung von $x_i(t+1)$ und $v_i(t+1)$

3.2 Ant Colony Optimization

Die von M. Dorigo[11] vorgestellte Ameisenkolonieoptimierung (ACO) dient, wie die Partikelschwarmoptimierung, der Lösung von Optimierungsproblemen. Zur näherungsweisen Lösung des Optimierungsproblems wird bei der ACO eine Population an modellierten Ameisen instanziiert und durch den vom Optimierungsproblem definierten Suchraum bewegt. Die Positionen der modellierten Ameisen repräsentieren potentielle Lösungen des Optimierungsproblems und werden in jedem Zeitschritt neu berechnet. Darüber hinaus ist jede Position im Suchraum mit einem Pheromonniveau markiert. Je höher das jeweilige Pheromonniveau, desto höher ist die Wahrscheinlichkeit, das sich die modellierten Ameisen in Richtung der mit Pheromonen markierten Positionen bewegen.

3.2.1 Suchraum

Der Suchraum $S \subset \mathbb{R}^n$ ist ein n-dimensionaler Raum, in dem sich die gesamte Population $X = \{x_1, x_2, ..., x_n\}$ der modellierten Ameisen bewegt. Damit eine Lösung gefunden werden kann beschränkt sich der Suchraum auf eine endliche Anzahl an möglichen Positionen $P = \{p_1, p_2, ..., p_n\}$ die von den modellierten Ameisen besucht werden können. Jeder Position P im Suchraum S ist ein Wert τ_i zugeordnet, der das Pheromonniveau der Position beschreibt.

3.2.2 Modellierte Ameise

Jede modellierte Ameise besitzt folgende Eigenschaften im Suchraum S zum Zeitpunkt t:

- **Position $x_i(t)$ im Suchraum S mit einem Funktionswert $f(x_i(t))$:** Die aktuelle Position der modellierten Ameise x_i zum Zeitpunkt t repräsentiert eine potentielle Lösung des Optimierungsproblems. Der dazugehörende Funktionswert $f(x_i(t))$ beschreibt die Güte der potentiellen Lösung.
- **Position $x_g(t)$ im Suchraum S mit einem Funktionswert $f(x_g(t))$:** Die Position x_g ist die bisher beste Position der Ameisenkolonie zum Zeitpunkt t.
- **Teillösungen $C = \{c_1(t), c_2(t), ..., c_n(t)\}$ die der modellierten Ameise x_i zum Zeitpunkt t zur Verfügung stehen:** Die modellierte Ameise x_i wählt aus der ihr zur Verfügung stehenden Teillösungen C zum Zeitpunkt t eine Teillösung. Die gewählte Teillösung entspricht der nächsten Position der modellierten Ameise x_i zum Zeitpunkt $t + 1$.
- **Wahrscheinlichkeiten $W = \{w_{c_1}(t), w_{c_2}(t), ..., w_{c_n}(t)\}$ zur Wahl von $x_i(t + 1)$:** Die Wahrscheinlichkeiten W werden für die, der modellierten Ameise zur Verfügung stehenden, Teillösungen C berechnet. Die Teillösung mit der höchsten Wahrscheinlichkeit wird von der modellierten Ameise ausgewählt.
- **Parameter α:** Der Parameter α bestimmt die Gewichtung der Pheromonmarkierungen der Teillösungen C bei der Berechnung der Wahrscheinlichkeiten W.
- **Parameter β:** Der Parameter β bestimmt die Gewichtung der Güte der Teillösungen C bei der Berechnung der Wahrscheinlichkeiten W

Mit $\alpha = 0$ und $\beta = 1$ erhält man einen Greedy-Algorithmus. Umgekehrt erhält man mit $\alpha = 1$ und $\beta = 0$ einen rein zufallsgesteuerten Algorithmus.

[11] M. Dorigo (1991)

3.2.3 Algorithmus

Initialisiere Ameisenkolonie X im Suchraum S;
while *Maximale Iterationen nicht erreicht* **do**
  Berechne für jede modellierte Ameise x_i die Wahrscheinlichkeiten W für die möglichen Teillösungen C ausgehend von Position $x_i(t)$;
  Bestimme für jede modellierte Ameise neue Position $x_i(t+1)$;
  if $f(x_i(t+1))$ *besser als* $f(x_g(t))$ **then** Setze $x_g(t))$ = $x_i(t+1)$;
  Berechne für jede gewählte Teillösung die neue Pheromonmarkierung $\tau_i(t+1)$ für Position P im Suchraum S;
end

Algorithm 2: ACO Algorithmus

Im ersten Schritt des abgebildeten Algorithmus wird die Population X an einer bestimmten Position P im Suchraum S initialisiert. Solange die Anzahl an maximalen Iterationen nicht erreicht ist berechnet jede Ameise die Wahrscheinlichkeiten W für die ihr zur Verfügung stehenden Teillösungen C mit der Formel:

$$W(c_i|x_i(t)) = \frac{\tau_{c_i}^{\alpha}(t) \cdot [f(c_i(t))]^{\beta}}{\sum c \in C\ \tau_c(t)^{\alpha} \cdot [f(c(t))]^{\beta}} \tag{1}$$

Die Wahrscheinlichkeit ist abhängig von dem Pheromonniveau tau_{c_i} der jeweiligen Teillösung und der Güte der Teillösung, bestimmt durch $f(c_i(t))$. Die Parameter α und β verändern das Verhalten der modellierten Ameisen. Gilt $\alpha > \beta$ dann ist das Pheromonniveau der Teillösung ausschlaggebender als die Güte der Teillösung für die Berechnung der Wahrscheinlichkeiten W. Sind die Wahrscheinlichkeiten W für alle Teillösungen C berechnet, entscheidet sich die modellierte Ameise für die Teillösung mit der höchsten Wahrscheinlichkeit. Die von der modellierten Ameise gewählte Teillösung entspricht der neuen Position $x_i(t+1)$. Im Anschluss wird für jede modellierte Ameise $x_i(t)$ geprüft, ob der aktuelle Funktionswert $f(x_i(t))$ besser ist als der globale Funktionswert $f(x_g(t))$. Ist dies der Fall so wird die global beste Position $x_g(t)$ zu $x_i(t)$. Am Ende der Iteration werden die Pheromonmarkierungen mit folgender Formel aktualisiert:

$$\tau_i(t+1) = (1-\rho)\tau_i(t) + \sum_{x_i \in X} \Delta\tau_i(t)^C \tag{2}$$

Hierbei ist ρ der konstante Verdunstungsfaktor der Pheromonmarkierungen. Die an der gewählten Teillösung hinterlassene Pheromonmenge $\Delta\tau_i(t)^C$ berechnet sich durch die Formel:

$$\Delta\tau_i(t)^C = f(c_i(t)) \tag{3}$$

3.3 Bee Colony Optimization

Die im Zeitraum 1999 bis 2003 von D. Teodorović und P. Lučić entwickelte Bienenkolonieoptimierung (BCO) stellt, wie die PSO und ACO, ein metaheuristisches Verfahren zur Lösung von Optimierungsproblemen dar. Die zur näherungsweisen Lösung des Optimierungsproblems, durch den Suchraum bewegten modellierten Bienen repräsentieren potentielle Lösungen des Optimierungsproblems, über ihre Position im Suchraum.

3.3.1 Suchraum

Der Suchraum S der BCO unterscheidet sich nicht von dem in Kapitel 3.1.1 beschriebenen Suchraum der PSO.

3.3.2 Modellierte Biene

Jede modellierte Biene besitzt folgende Eigenschaften im Suchraum S zum Zeitpunkt t:

- **Position $x_i(t)$ im Suchraum S mit einem Funktionswert $f(x_i(t))$:** Die aktuelle Position der modellierten Biene x_i zum Zeitpunkt t repräsentiert eine potentielle Lösung des Optimierungsproblems. Der dazugehörende Funktionswert $f(x_i(t))$ beschreibt die Güte der potentiellen Lösung.
- **Position $x_g(t)$ im Suchraum S mit einem Funktionswert $f(x_g(t))$:** Die Position x_g ist die bisher beste Position der Bienenkolonie zum Zeitpunkt t.
- **Loyalitätswahrscheinlichkeit $l_i(t)$:** Die Loyalitätswahrscheinlichkeit $l_i(t)$ der modellierten Biene x_i bestimmt den Status Z zum Zeitpunkt t.
- **Status Z:** Ist die Loyalitätswahrscheinlichkeit $l_i(t)$ größer als eine Zufallszahl so wird die modellierte Biene x_i zum Zeitpunkt t in den Status Loyal versetzt. Die modellierte Biene erhält den Status Frei sobald die Loyalitätswahrscheinlichkeit kleiner ist als die Zufallszahl.
- **(optional) Rekrutierungsbienen $R = \{r_1(t), r_2(t), ..., r_n(t)\}$ die der modellierten Biene x_i zum Zeitpunkt t zur Verfügung stehen:** Die modellierte Biene x_i wählt aus der ihr zur Verfügung stehenden Rekrutierungsbienen R zum Zeitpunkt t eine Rekrutierungsbiene.
- **(optional) Richtung $v_i(t)$:** Die Richtung $v_i(t)$ von der aktuellen Position $x_i(t)$ der modellierten Biene zur gewählten Rekrutierungsbiene $r_i(t)$.

Befindet sich eine modellierte Biene x_i im Status Loyal besitzt sie zum Zeitpunkt t keine Rekrutierungsbienen R sondern ist ihrer Lösung $x_i(t)$ gegenüber loyal und wird diese nicht verändern. Ebenso besitzt sie daher keine Richtung $v_i(t)$.

3.3.3 Algorithmus

Initialisiere Bienenkolonie X im Suchraum S;
while *Maximale Iterationen nicht erreicht* **do**
  Bestimme für jede modellierte Biene Position $x_i(t)$ und berechne den Funktionswert $f(x_i(t))$;
  Bestimme n Rekrutierungsbienen R mit den besten Funktionswerten $f(x_i(t))$;
  Berechne für jede modellierte Biene die Loyalitätswahrscheinlichkeit $l_i(t)$;
  if $l_i(t)$ *größer als Zufallszahl* **then**
    Versetze Biene in den Status Loyal;
  else
    Versetze Biene in den Status Frei;
  end
  if *Biene im Zustand Frei* **then**
    Berechne Rekrutierungswahrscheinlichkeiten W für Rekrutierungsbienen R und wähle Rekrutierungsbiene $r_i(t)$;
    Bestimme Richtung $v_i(t)$ von Position $x_i(t)$ zu $r_i(t)$;
    Setze neue Position $x_i(t+1)$ = $x_i(t)$ + $v_i(t)$;
  else
    Setze neue Position $x_i(t+1)$ = $x_i(t)$;
  end
  if $f(x_i(t+1))$ *besser als* $f(x_g(t))$ **then** Setze $x_g(t))$ = $x_i(t+1)$;
end

Algorithm 3: BCO Algorithmus

$$l_i(t) = e^{-\frac{f(x_g(t)) - f(x_i(t))}{u}} \tag{1}$$

$$r_i(t) = \frac{f(r_i(t)}{\sum_{r \in R} f(r(t))} \tag{2}$$

4 Modelle und Frameworks

4.1 Modelle

Die in Kapitel 3 vorgestellten Algorithmen sollen in agentenbasierten Modellen umgesetzt werden und Anwendung finden. Als abstraktes Beispiel wird die PSO angeführt. Die PSO wird dazu verwendet das globale Minimum der Rastrigin-Funktion zu finden. Die ACO und BCO werden als Strategie verwendet um die Futtersuche von Ameisen und Bienen zu modellieren.

4.1.1 Finden des globalen Minimums einer Funktion mit PSO

Modellbeschreibung Im Modell werden interagierende Agenten durch die zweidimensionale Agenten-Umgebung bewegt um das globale Minimum der, in Abbildung 3 abgebildeten, zweidimensionalen Rastrigin-Funktion zu finden. Die Rastrigin-Funktion wurde 1974 von A. Leonard vorgeschlagen und dient der Performanceanalyse von Optimierungsalgorithmen. Die Rastrigin-Funktion ist definiert durch:

$$f(x) = An + \sum_{i=0}^{n} [x_i^2 - A\cos(2\pi x_i)] \tag{1}$$

Dabei ist $A = 10$ eine Konstante, n die Dimensionen. Außerdem gilt $x = x_1, ..., x_n$ mit $x_i \in [-5.12, 5.12]$.

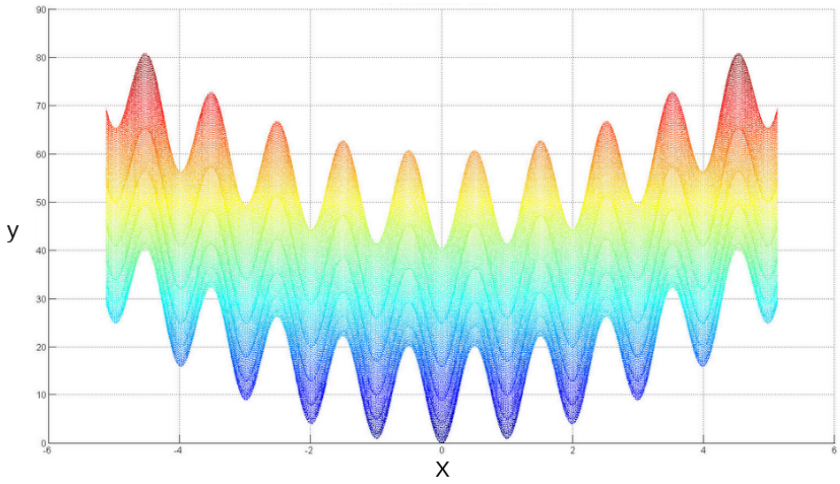

Abbildung 3: Rastrigin-Funktion

Methode Das Modell zur Simulation der PSO ist in dieser Arbeit als abstraktes Beispiel zur agentenbasierten Modellierung angeführt. Die Methode zum Finden des globalen Minimums der Rastrigin-Funktion stellt der im Kapitel 3.3 vorgestellte PSO-Algorithmus dar.

Parameter Im Modell ist die Anzahl der Partikel, der Trägheitskoeffizient, der lokale Vertrauenskoeffizient und der globale Vertrauenskoeffizient einstellbar.

4.1.2 Futtersuche mit ACO und PSO

Modellbeschreibung Im Modell der Futtersuche suchen die Agenten nach der Futterquelle in einer zweidimensionalen Agenten-Umgebung. Sie starten ihre Suche im Ameisenbau oder Bienenstock. Das erste Ziel des Modells ist das zufallsgesteuerte Finden der Futterquelle durch die Agenten. Im Anschluss interagieren die Agenten miteinander um die Futterquelle abzubauen.

MethodeACO Zum effizienten Abbau der Futterquelle kommunizieren die modellierten Ameisen über die zweidimensionale Agenten-Umgebung in Form von Pheromonmarkierungen. Im Modell gibt es zwei Pheromontypen die mit der Zeit verdunsten. Heimatpheromone markieren den Weg zum Ameisenbau und Futterpheromone den Weg zur Futterquelle. Auf der Suche nach der Futterquelle werden Heimatpheromone hinterlassen. Findet eine modellierte Ameise die Futterquelle wird eine Einheit der Futterquelle abgebaut. Nachdem die Futterquelle abgebaut wurde kehrt die Ameise zum Ameisenbau zurück. Dabei hinterlässt sie Futterpheromone. Befindet sich die modellierte Ameise im Ameisenbau liefert sie die Futtereinheit und begibt sich anschließend erneut auf Futtersuche. Der abgebildete Algorithmus 4 fasst das Vorgehen für den Abbau der Futterquelle zusammen.

if *hatFuttereinheit* **then**
  Kehre zum Ameisenbau zurück (Folge Heimatpheromonen);
  Hinterlasse Futterpheromone;
  if *Position der Ameise == Position des Ameisenbaus* **then** hatFuttereinheit = false ;
else
  Finde Futterquelle (Folge Futterpheromonen);
  Hinterlasse Heimatpheromone;
  if *Position der Ameise == Position der Futterquelle* **then** hatFuttereinheit = true ;
end

Algorithm 4: Futtersuche mit ACO

Parameter ACO Die einstellbaren Parameter des Modells sind die Anzahl der modellierten Ameisen und die in Kapitel 3.2.2 definierten Parameter α und β der ACO.

```
if InformationFutterquelle then
    Bewege dich in die Richtung der Futterquelle;
    Finde die Futterquelle;
    Kehre zum Bienenstock zurück;
    Tanze für aufgeschlossene Bienen;
else
    if Tanzende Biene in der Umgebung then
        Beobachte tanzende Bienen;
        InformationFutterquelle = true;
    else
        Starte mit niedriger Wahrscheinlichkeit eine zufällige Suche nach der
          Futterquelle;
        if Suchlimit then
            Kehre zum Bienenstock zurück;
        else
            Finde die Futterquelle;
            InformationFutterquelle = true;
            Kehre zum Bienenstock zurück;
            Tanze für aufgeschlossene Bienen;
        end
    end
end
```

Algorithm 5: Futtersuche mit ACO

Methode BCO

Parameter BCO

4.2 Framework

4.2.1 Anforderungen

Die Anforderungen an das Framework zur Erstellung der in Kapitel 4.1 vorgestellten Modelle und anschließender grafischer Simulation sind:

- **Open-Source + kostenlos:** Der Quelltext des Frameworks ist öffentlich. Zudem kann das Framework kostenlos genutzt werden.
- **Aktualität:** Es ist eine aktuelle und stabile Version des Frameworks verfügbar.
- **Einsetzbarkeit:** Das Framework kann auf dem Betriebssystem Linux verwendet werden und unterstützt die Programmiersprache Java. Diese Anforderung entsteht durch die Präferenzen des Autors dieser Arbeit.
- **Visualisierung:** Modelle werden grafisch in 2D simuliert.
- **Dokumentation + Tutorial:** Das Framework ist umfangreich dokumentiert und besitzt ein Tutorial für Einsteiger.
- **Ausführungsgeschwindigkeit der Simulation:** Die Ausführungsgeschwindigkeit der Simulation ist hoch, trotz rechenintensiven Modellen mit vielen Agenten und Iterationen.

4.2.2 Übersicht

NetLogo NetLogo ist eine Open-Source-Plattform und wurde 1999 von U. Wilensky entwickelt. Die neuste Version 6.0.3 wurde im März 2018 veröffentlicht und steht für die Betriebssysteme Mac, Windows und Linux zur Verfügung. Die proprietäre funktionale Programmiersprache und die ausführliche Dokumentation machen es Einsteigern ohne Programmierkenntnisse einfach erste eigene Modelle zu implementieren. Das Fehlen objektorientierter Features schränkt die Funktionalität und Erweiterbarkeit des Frameworks jedoch ein. Darüber hinaus verfügt NetLogo über keine explizite Unterstützung für die Trennung zwischen Agenten, Agenten-Beziehungen und Agenten-Umgebung.

Mason Mason ist eine Java-Bibliothek, die ein breitgefächertes Repertoire an Funktionen für die ABM bereitstellt. Die aktuelle Version 1.9 wurde im Juni 2015 veröffentlicht. Eine umfangreiche Dokumentation der Open-Source-Bibliothek und ein einsteigerfreundliches Tutorial erlauben es innerhalb weniger Stunden eine erste lauffähige Simulation zu erstellen. Kombiniert mit der JFreeChart-Bibliothek ist es zudem möglich die Simulationen in grafischer Form auszuwerten.

Swarm Swarm ist eine der ersten ABM-Plattformen. Ursprünglich wurde Swarm 1997 am Santa Fe Institute entwickelt und veröffentlicht. Die ABM-Plattform Swarm war zuerst nur auf UNIX-basierten Betriebssystemen einsetzbar und unterstützte nur die Programmiersprache Objective-C. Heute steht Swarm für die Betriebssysteme Mac, Windows und Linux zur Verfügung und unterstützt neben Objective-C auch Java. Das Framework ist Open-Source. Version 2.4.1 wurde im April 2009 veröffentlicht.

Repast Repast ist eine Open-Source-Plattform für die ABM und Simulation. Die Software ist für die Betriebssysteme Mac, Windows und Linux verfügbar. Es existieren zwei verschiedene Editionen der Plattform. Repast Simphony implementiert grundlegende und erweiterte Konzepte der ABM. Unterstützt werden die Programmiersprachen Java, Logo und Groovy. Repast High Performance Computing ist ein agentenbasiertes Modellierungssystem speziell entwickelt für die parallele Ausführung der Simulationen auf verteilten Systemen. Zur Modellentwicklung mit

dieser Edition von Repast werden umfangreiche Programmierkenntnisse in C++ vorausgesetzt. Die stabile Version 2.5 von Repast Symphonie wurde im Oktober 2017 veröffentlicht.

Zusammenfassung S.F. Railsback, S.L. Lytinen und S.K. Jackson untersuchten im Jahr 2006 verschiedene Plattformen für die ABM und Simulation. Ihre Ergebnisse über die oben genannten ABM-Plattformen können folgendermaßen zusammengefasst werden:

- **NetLogo:** Netlogo ist einfach zu bedienen und besitzt eine umfangreiche Dokumentation. NetLogo ist eine gute Wahl für simple Modelle bei denen die Ausführungszeit der Simulation eine untergeordnete Rolle spielt. Außerdem eignet sich NetLogo zur Erstellung eines Modellprototyps. Dieser Prototyp kann anschließend in anderen ABM-Plattformen erweitert werden.
- **Mason:** Mason ist eine gute Wahl für erfahrene Programmierer, die an rechenintensiven Modellen mit vielen Agenten und Iterationen arbeiten. Ein Vorteil von Mason ist die Reproduzierbarkeit der Simulationsergebnisse auf verschiedenen Betriebssystemen.
- **Swarm:** Swarm ist stabil, klein, gut organisiert und unterstützt komplexe Modelle. Voraussetzung für einen schnellen Einstieg ist Programmiererfahrung mit Objective-C.
- **Repast:** Repast ist die vollständigste ABM-Plattform und hat eine vergleichsweise schnelle Ausführungszeit der Simulationen. Die Dokumentation ist teilweise unvollständig.

4.2.3 Auswahl

Tabelle 1 zeigt eine Übersicht der Frameworks und die erfüllten Anforderungen. Zur Implementierung der in Kapitel 4.1 beschriebenen Modelle wird Mason als Framework gewählt. Mit Mason können die Modelle in Java implementiert werden. Da im Zuge dieser Studienarbeit rechenintensive Simulationen mit vielen Agenten durchgeführt werden sollen ist die hohe Ausführungsgeschwindigkeit der Simulationen wichtig. Mason bietet zudem eine umfangreiche Dokumentation und sowie ein 14-stufiges Tutorial.

	Open-Source + kostenlos	Aktualität	Einsetzbarkeit	Visualisierung	Doku + Tutorial	Geschwindigkeit
NetLogo	$\surd$	Mrz. 2018	X	2D/3D	$\surd$	langsam
Mason	$\surd$	Jun. 2015	$\surd$	2D/3D	$\surd$	sehr schnell
Swarm	$\surd$	Apr. 2009	$\surd$	2D/3D	X	langsam
Repast	$\surd$	Okt. 2017	$\surd$	2D/3D	$\surd$	schnell

Tabelle 1: Auswahl des Frameworks

5 Implementierung

5.1 Mason

5.2 Finden des globalen Minimums einer Funktion mit PSO

Bei der Implementierung des Modells wird ein Großteil der Funktionen des Modells von A. Desai, S. Luke und G. Mason verwendet. Ihr Modell darf modifiziert und veröffentlicht werden. Um die Implementierung des in Kapitel 4.1.1 definierten Modells zu beschreiben wird die Implementierung anhand der Schritte des PSO-Algorithmus aus Kapitel 3.1.3 beschrieben.

5.2.1 Initalisiere Partikelschwarm im Suchraum

```
public void start(){
  bestVal = 0;
  super.start();
  particles = new Particle[numParticles];
  space = new Continuous2D(height, width, height);
  Evaluatable f = mapFitnessFunction(fitnessFunction);
  //Initalisiere Partikel
  for (int i = 0; i < numParticles; i++){
    double x = (random.nextDouble() * width) - (width * 0.5);
    double y = (random.nextDouble() * height) - (height * 0.5);
    double vx = (random.nextDouble() * initialVelocityRange) - (
        initialVelocityRange * 0.5);
    double vy = (random.nextDouble() * initialVelocityRange) - (
        initialVelocityRange * 0.5);

    final Particle p = new Particle(x, y, vx, vy, this, f, i);
    particles[i] = p;
  }
}
```

Listing 1: xxxxxxx

5.2.2 Bestimme Position und Geschwindigkeit

```
public Particle(double x, double y, double vx, double vy, PSO pso, Evaluatable
    f) {
  super();
  //Bestimme Position und Geschwindigkeit der Partikel
  this.position.setTo(x, y);
  this.velocity.setTo(vx, vy);

  this.pso = pso;
  this.fitnessFunction = f;
  pso.space.setObjectLocation(this,new Double2D(position));
}
```

Listing 2: xxxxxxx

5.2.3 Bestimme den Fitnesswert

```
public class Rastrigin implements Evaluatable{
  private static final long serialVersionUID = 1;

  public double calcFitness(double x, double y){
    return (1000 - (20 + x*x - 10*Math.cos(2*Math.PI*x) + y*y - 10*Math.cos(2*
        Math.PI*y)));
```

```
  }
}
```

Listing 3: xxxxxxx

5.2.4 Bestimme beste Position

```
public void updateBest(double currVal, double currX, double currY){
  if (currVal > bestVal){
    bestVal = currVal;
    bestPosition.setTo(currX, currY);
    pso.updateBest(currVal, currX, currY);
  }
}
```

Listing 4: xxxxxxx

5.2.5 Berechne neue Position

```
public void stepUpdatePosition(){
  position.addIn(velocity);
  pso.space.setObjectLocation(this, new Double2D(position));
}
//Defintion von addIn
public final MutableDouble2D addIn(final MutableDouble2D other){
  x = other.x + x;
  y = other.y + y;
  return this;
}
```

Listing 5: xxxxxxx

5.2.6 Berechne neue Geschwindigkeit

```
public void stepUpdateVelocity(){

  double x = position.x;
  double y = position.y;
  double inertia = velocity.x;
  double pDelta = bestPosition.x - x;
  double gDelta = pso.bestPosition.x - x;
  double pWeight = Math.random() + 0.4;
  double gWeight = Math.random() + 0.4;
  double vx = (0.9*inertia + pWeight*pDelta + gWeight*gDelta) / (1+pWeight+
      nWeight+gWeight);

  inertia = velocity.y;
  pDelta = bestPosition.y - y;
  gDelta = pso.bestPosition.y - y;
  pWeight = Math.random() + 0.4;
  gWeight = Math.random() + 0.4;
  double vy = (0.9*inertia + pWeight*pDelta + gWeight*gDelta) / (1+pWeight+
      nWeight+gWeight);

  velocity.setTo(vx, vy);
}
```

Listing 6: xxxxxxx

Literatur

[1] A. Aulinger. Kollektive Intelligenz: Methoden, Erfahrungen und Perspektiven. Entstehungsjahr: 2009.

[2] M. Dorigo, V. Maniezzo, A. Colorni. Ant System: An Autocatalytic Optimizing Process. Technical Report 91-016 Revised. Entstehungsjahr: 1991.

[3] Y. Liu. Partikelschwarmoptimierung für diskrete Probleme. Technische Universität München. Entstehungsjahr: 2014.

Weblink: http://lydiapintscher.de/uni/schwarmintelligenz.pdf
Einsichtnahme: 14.01.2018

[4] C. Macal, M. North. Tutorial on agent-based modelling and simulation. Journal of Simulation 4, Nr. 1, Seite 151-162. Entstehungsjahr: 2010.

[5] L. Pintscher. Schwarmintelligenz. Universität Karlsruhe. Entstehungsjahr: ohne Jahr.

Weblink: http://lydiapintscher.de/uni/schwarmintelligenz.pdf
Einsichtnahme: 14.01.2018

[6] S. Russel, P. Norvig. Artificial Intelligence: A Modern Approach. Upper Saddle River. Entstehungsjahr: 2009.

[7] W.M. Wheeler. The Ant Colony as an Organism. Journal of Morphology Volume 22, Issue 2, Seite 307-325. Entstehungsjahr: 1911.

Weblink: http://www3.interscience.wiley.com/journal/109914213/abstract
Einsichtnahme: 04.01.2018

[8] M. Wooldridge, N. Jennings. Intelligent Agents: Theory and Practice. Knowledge Engineering Review 10, Nr. 2, Seite 115-152. Entstehungsjahr: 1995.

Verzeichnis der Anhänge

8 Anhang

8.1 Quellcode Futtersuche mit ACO

8.1.1 Kehre zum Ameisenbau zurück + (Folge Heimatpheromonen)

```
double max = AntsForage.IMPOSSIBLY_BAD_PHEROMONE;

int max_x = x;
int max_y = y;
int count = 2;

for(int dx = -1; dx < 2; dx++) {
for(int dy = -1; dy < 2; dy++) {

int _x = dx+x;
int _y = dy+y;

if ((dx == 0 && dy == 0) || _x < 0 || _y < 0 || _x >= AntsForage.GRID_WIDTH ||
    _y >= AntsForage.GRID_HEIGHT ||  af.obstacles.field[_x][_y] == 1) continue;

double m = af.toHomeGrid.field[_x][_y];


if (m > max) {count = 2;}

if (m > max || (m == max && state.random.nextBoolean(1.0 / count++))) {
max = m;
max_x = _x;
max_y = _y;
}
}
}

if (max == 0 && last != null) {

if (state.random.nextBoolean(af.momentumProbability)) {
int xm = x + (x - last.x);
int ym = y + (y - last.y);

if (xm >= 0 && xm < AntsForage.GRID_WIDTH && ym >= 0 && ym < AntsForage.
    GRID_HEIGHT && af.obstacles.field[xm][ym] == 0) {
max_x = xm;
max_y = ym;
}
}
} else if (state.random.nextBoolean(af.randomActionProbability)) {
int xd = (state.random.nextInt(3) - 1);
int yd = (state.random.nextInt(3) - 1);
int xm = x + xd;
int ym = y + yd;

if (!(xd == 0 && yd == 0) && xm >= 0 && xm < AntsForage.GRID_WIDTH && ym >= 0
    && ym < AntsForage.GRID_HEIGHT && af.obstacles.field[xm][ym] == 0) {
max_x = xm;
max_y = ym;
}
}

af.buggrid.setObjectLocation(this, new Int2D(max_x, max_y));

if (af.sites.field[max_x][max_y] == AntsForage.HOME) {
hasFoodItem = ! hasFoodItem;
```

```
55 }
```

Listing 7: xxxxxxx